Making Indefinite Kernel Learning Practical

Ingo Mierswa
Artificial Intelligence Unit
Department of Computer Science
University of Dortmund
ingo.mierswa@uni-dortmund.de

Abstract

In this paper we embed evolutionary computation into statistical learning theory. First, we outline the connection between large margin optimization and statistical learning and see why this paradigm is successful for many pattern recognition problems. We then embed evolutionary computation into the most prominent representative of this class of learning methods, namely into Support Vector Machines (SVM). In contrast to former applications of evolutionary algorithms to SVM we do not only optimize the method or kernel parameters. We rather use evolution strategies in order to directly solve the posed constrained optimization problem. Transforming the problem into the Wolfe dual reduces the total runtime and allows the usage of kernel functions just as for traditional SVM. We will show that evolutionary SVM are at least as accurate as their quadratic programming counterparts on eight real-world benchmark data sets in terms of generalization performance. They always outperform traditional approaches in terms of the original optimization problem. Additionally, the proposed algorithm is more generic than existing traditional solutions since it will also work for non-positive semidefinite or indefinite kernel functions. The evolutionary SVM variants frequently outperform their quadratic programming competitors in cases where such an indefinite Kernel function is used.

1 Introduction

In this paper we will discuss how evolutionary algorithms can be used to solve large margin optimization problems. We explore the intersection of three

highly active research areas, namely machine learning, statistical learning theory, and evolutionary algorithms. While the connection between statistical learning and machine learning was analyzed before, embedding evolutionary algorithms into this connection will lead to a more generic algorithm which can deal with problems today's learning schemes cannot cope with.

Supervised machine learning is often about classification problems. A set of data points is divided into several classes and the machine learning method should learn a decision function in order to decide into which class an unseen data point should be classified.

The maximization of a margin between data points of different classes, i. e. the distance between a decision hyperplane and the nearest data points, interferes with the ideas of statistical learning theory. This allows the definition of an error bound for the generalization error. Furthermore, the usage of kernel functions allows the learning of non-linear decision functions. We focus on Support Vector Machines (SVM) as they are the most prominent representatives for large margin problems. Since SVM guarantee an optimal solution for the given data set they are currently one of the mostly used learning methods. Furthermore, many other optimization problems can also be formulated as large margin problem [34]. The relevance of large margin methods can be measured by the number of submissions to the main machine learning conferences over the past years[1].

Usually, the optimization problem posed by SVM is solved with quadratic programming. However, there are some drawbacks. First, for kernel functions which are not positive semidefinite no unique global optimum exists. In these cases quadratic programming is not able to find satisfying solutions at all. Moreover, most implementations do not even terminate [8]. There exist several useful non-positive kernels [15], among them the sigmoid kernel which simulates a neural network [4, 31]. A more generic optimization scheme should allow such non-positive kernels without the need for omitting the more efficient dual optimization problem [22].

Former applications of evolutionary algorithms to SVM include the optimization of method and kernel parameters [6, 27], the selection of optimal feature subsets [7], and the creation of new kernel functions by means of genetic programming [10]. The latter is particularly interesting since it cannot be guaranteed that the resulting kernel functions are again positive semidefinite.

Replacing the traditional optimization techniques by evolution strategies or particle swarm optimization can tackle the problems mentioned above. We

[1]More than 30% of all accepted papers for ICML 2005 dealt with SVM and other large margin methods.

will extract as much information as possible from the optimization problem at hand and develop and compare different search point operations. First, we will show that the proposed implementation leads to as good results as traditional SVM on a broad variety of real-world benchmark data sets. Additionally, the optimization is more generic since it also allows for non-positive semidefinite kernel functions.

1.1 Outline

In Section 2 we give a short introduction into the concept of regularized risk minimization and the ideas of statistical learning theory. We will also discuss an upper bound for the generalization error. This allows us to formalize the optimization problem of large margin methods in Section 3. We will introduce SVM for the classification of given data points in Section 3.1 and extend the separation problem to non-separable data sets (see Section 3.2) with non-linear hyperplanes (see Section 3.3). In Section 4 we will discuss the effect of non-positive semidefinite Kernels and former strategies to handle the problems introduced by this type of base functions. The constrained optimization problem developed in the previous sections will be solved by evolution strategies and particle swarm optimization which is described in Section 5. We discuss several enhancements and a new type of mutation before we evaluate the proposed methods on synthetical and real-world benchmark data sets in Section 6.

2 Regularized Risk Minimization

In this section, we discuss the idea of regularized risk minimization. Machine learning methods following this paradigm have a solid theoretical foundation and it is possible to define bounds for prediction errors.

Let the instance space be defined as Cartesian product $X = X_1 \times \ldots \times X_m$ of attributes $X_i \subseteq \mathbb{R}$. Let Y be another set of possible labels. X and Y are random variables obeying a fixed but unknown probability distribution $P(X, Y)$. *Machine Learning* tries to find a function $f(x, \gamma)$ which predict the value of Y for a given input $x \in X$. The function class f depends on a vector of parameters γ, e. g. if f is the class of all polynomials, γ might be the degree. We define a *loss function* $L(Y, f(X, \gamma))$ in order to penalize errors during prediction [9]. Every convex function with arity 2, positive range, and $L(x, x) = 0$ can be used as loss function [30]. This leads to a possible criterion for the selection of a function f, the *expected risk*:

Definition 1 *Let X be a vector of random variables and Y another random variable obeying a fixed but unknown probability distribution $P(X, Y)$. For a loss function $L(Y, f(X, \gamma))$ the* EXPECTED RISK *is defined as*

$$R(\gamma) = \int L(y, f(x, \gamma)) dP(x, y).$$

Since the underlying distribution is not known we are not able to calculate the expected risk. However, instead of estimating the probability distribution in order to allow this calculation, we directly estimate the expected risk by using a set of known data points $T = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq X \times Y$. T is usually called *training data*. Using this set of data points we can calculate the *empirical risk*:

Definition 2 *Let $T = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq X \times Y$ be an item set and let $L(Y, f(X, \gamma))$ be a a loss function. The* EMPIRICAL RISK *is defined as*

$$R_{emp}(\gamma) = \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, f\left(x_i, \gamma\right)\right).$$

If training data is sampled according to $P(X, Y)$, the empirical risk approximates the expected risk if the number of samples grows:

$$\lim_{n \to \infty} R_{emp}(\gamma) = R(\gamma).$$

It is, however, a well known problem that for a finite number of samples the minimization of $R_{emp}(\gamma)$ alone does not lead to a good prediction model [36]. For each loss function L, each candidate γ, and each set of tuples $T' \subseteq X \times Y$ with $T \cap T' = \emptyset$ exists another parameter vector γ' so that $L(y, f(x, \gamma)) = L(y, f(x, \gamma'))$ for all $x \in T$ and $L(y, f(x, \gamma)) > L(y, f(x, \gamma'))$ for all $x \in T'$. Therefore, the minimization of $R_{emp}(\gamma)$ alone does not guarantee the optimal selection of a parameter vector γ for other samples according to the distribution $P(X, Y)$. This problem is often referred to as overfitting.

At this point we use one of the main ideas of statistical learning theory. Think of two different functions perfectly approximating a given set of training points. The first function is a linear function, i. e. a simple hyperplane in the considered space $\mathbb{R}^m$. The second function also hits all training points but is strongly wriggling in between. Naturally, if we had to choose between these two approximation functions, we tend to select the more simple one, i. e. the linear hyperplane in this example. This derives from the observation that more simple functions behave better on unseen examples than very complicated functions. Since the mere minimization of the empirical risk

according to the training data is not appropriate to find a good generalization, we incorporate the *capacity*[2] of the used function into the optimization problem (see Figure 1). This leads to the minimization of the *regularized risk*:

Definition 3 *Let Ω be strictly monotonic increasing function. The* REGULARIZED RISK *is defined as*

$$R_{reg}(\gamma) = R_{emp}(\gamma) + \lambda\Omega(\gamma).$$

This risk functional is also known as *structural risk* since it takes the structural complexity into account. Ω is a function which measures the capacity of the function class f depending on the parameter vector γ (see [29] for more details). Since the empirical risk is usually a monotonically decreasing function of Ω, we use λ to manage the trade-off between training error and capacity. Methods minimizing this type of risk function are known as *shrinkage estimators* [11]. We refer to the idea of taking the structural complexity into account as *regularization* and to λ as *regularization parameter*.

2.1 Bound on the Generalization Performance

For certain functions Ω, the regularized risk is an upper bound for the empirical risk. The capacity of the function f for a given γ can for example be measured with help of the *Vapnik-Chervonenkis dimension* (VC dimension) [36, 37]. The VC dimension is defined as the cardinality of the biggest set of tuples which can separated with help of f in all possible ways. For example, the VC dimension of linear hyperplanes in an m-dimensional space is $m+1$. Using the VC dimension as a measure for capacity leads to a probabilistic bound for the regularized risk [36]. Let f be a function class with finite VC dimension h and $f(\gamma)$ the best solution for the empirical risk minimization for T with $|T| = n$. Now choose some η such that $0 \leq \eta \leq 1$. Then for losses smaller than some number B, the following bound holds with probability $1 - \eta$:

$$R(\gamma) \leq R_{emp}(\gamma) + B\sqrt{\frac{h\left(\log\frac{2l}{h} + 1\right) - \log\frac{\eta}{4}}{l}}.$$

Surprisingly, this bound is independent of $P(X, Y)$. It only assumes that both, the seen and the unseen data points, are independently sampled according to some $P(X, Y)$. Please note that this bound also no longer contains

[2] Although not the same, the capacity of a function resembles a measurement of the function complexity. In our example we measure the ability to "wriggle". More details in [36].

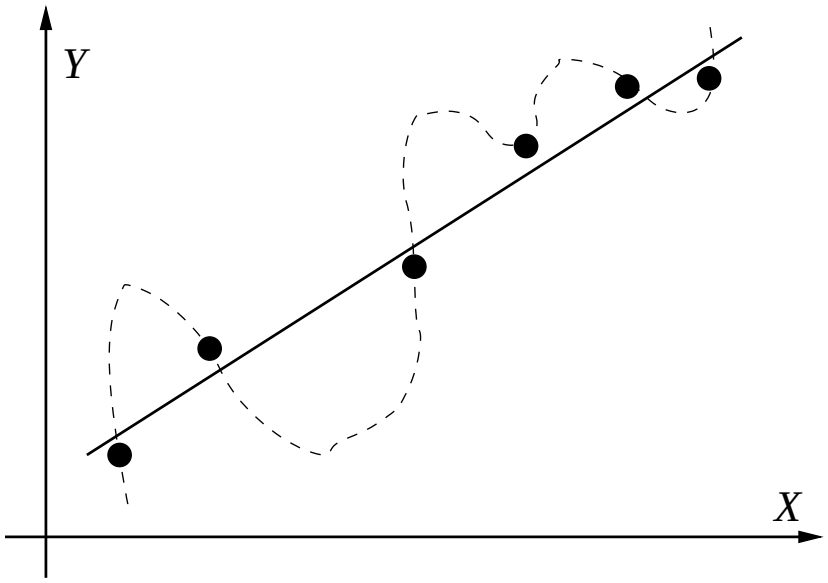

Figure 1: The simultaneous minimization of empirical risk and model complexity gives a hint which function should be used in order to generalize the given data points.

a weighting factor λ or any other trade-off at all. The existence of a guaranteed error bound is the reason for the great success of regularized risk minimization in a wide range of applications.

3 Large Margin Methods

As discussed in the previous section, we need to use a class of functions whose capacity can be controlled. In this section we will discuss a special form of regularized risk minimization, namely large margin approaches. All large margin methods have one thing in common: they embed regularized risk minimization by maximizing a margin between a linear function and the nearest data points. The most prominent large margin method for classification tasks is the *Support Vector Machine* (SVM).

3.1 Support Vector Machines

We constrain the number of possible values of Y to 2, without loss of generality these values should be -1 and $+1$. In this case, finding a function f in order to decide which of both predictions is correct for an unseen data point is referred to as *classification learning* for the classes -1 and $+1$. We start

with the simplest case: learning a linear function from perfectly separable data. As we shall see in Section 3.2 and 3.3, the general case - non-linear functions derived from non-separable data - leads to a very similar problem.

If the data points are linearly separable, a linear hyperplane must exist in the input space $\mathbb{R}^m$ which separates both classes.

Definition 4 *A* SEPARATING HYPERPLANE *is defined as*

$$H = \{x | \langle w, x \rangle + b = 0\},$$

where w is normal to the hyperplane, $|b|/||w||$ is the perpendicular distance of the hyperplane to the origin (offset or bias), and $||w||$ is the Euclidean norm of w.

The vector w and the offset b define the position and orientation of the hyperplane in the input space. These parameters correspond to the function parameters γ discussed above. After the optimal parameters w and b were found, the prediction of new data points can be calculated as

$$f(x, w, b) = \text{sgn}\left(\langle w, x \rangle + b\right),$$

which is one of the reasons why we constrained the classes to -1 and $+1$.

Figure 2 shows some data points and a separating hyperplane. If all given data points are correctly classified by the hyperplane the following must hold:

$$\forall i : y_i \left(\langle w, x_i \rangle + b\right) \geq 0. \tag{1}$$

Of course, an infinite number of different hyperplanes exist which perfectly separate the given data points. However, one would intuitively choose the hyperplane which has the biggest amount of safety margin to both sides of the data points. Normalizing w and b in a way that the point(s) closest to the hyperplane satisfy $|\langle w, x_i \rangle + b| = 1$ we can transform equation (1) into

$$\forall i : y_i \left(\langle w, x_i \rangle + b\right) \geq 1.$$

We can now define the *margin* as the perpendicular distance of the nearest point(s) to the hyperplane. Consider two points x_1 and x_2 on opposite sides of the margin. That is $\langle w, x_1 \rangle + b = +1$ and $\langle w, x_2 \rangle + b = -1$ and $\langle w, (x_1 - x_2) \rangle = 2$. The margin is then given by $1/||w||$.

It can be shown that the capacity of the class of separating hyperplanes decreases with increasing margin [29]. Maximizing the margin of a hyperplane therefore formalizes the regularized risk minimization discussed in the previous section (regularization). Instead of maximizing $1/||w||$ we could also minimize $\frac{1}{2}||w||^2$ which will result into more simple equations later. This leads to the following optimization problem:

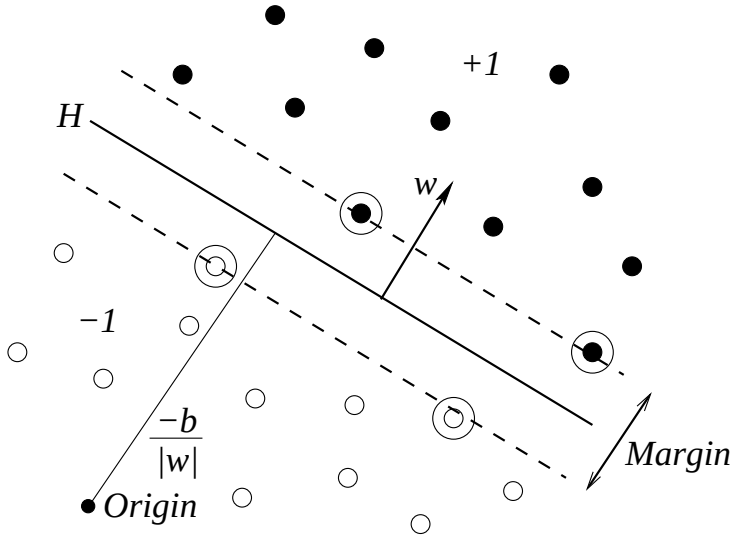

Figure 2: A simple binary classification problem for two classes -1 (empty bullets) and $+1$ (filled bullets). The separating hyperplane is defined by the vector w and the offset b. The distance between the nearest data point(s) and the hyperplane is called *margin*.

Problem 1 *The SVM problem for linearly separable data is defined as*

$$\textit{minimize } \tfrac{1}{2}||w||^2 \tag{2}$$

$$\textit{subject to } \forall i : y_i \left(\langle w, x_i \rangle + b\right) \geq 1. \tag{3}$$

Function (2) is the original *objective function* and the constraints from equation (3) are called *inequality constraints*. They form a *constrained optimization problem*. We will use a Lagrangian formulation of the problem. This allows us to replace the inequality constraints by constraints on the Lagrange multipliers which are easier to handle. The second reason is that after the transformation of the optimization problem, the training data will only appear in dot products. This will allow us to generalize the optimization to the non-linear case (see Section 3.3). We will now introduce positive Lagrange multipliers $\alpha_i, i = 1, \ldots, n$, one for each of the inequality constraints. The Lagrangian has the form

$$L_P(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i y_i \left(\langle w, x_i \rangle + b\right). \tag{4}$$

Finding a minimum of this function requires that the derivatives

$$\frac{\partial L_P(w,b,\alpha)}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i$$
$$\frac{\partial L_P(w,b,\alpha)}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i$$

are zero, i. e.

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{5}$$
$$0 = \sum_{i=1}^{n} \alpha_i y_i. \tag{6}$$

The Wolfe dual, which has to be maximized, results from the Lagrangian by substituting (5) and (6) into (4), thus

$$L_D(w,b,\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle .$$

This leads to the dual optimization problem which must be solved in order to find a separating maximum margin hyperplane for given set of data points:

Problem 2 *The dual SVM problem for linearly separable data is defined as*

$$\begin{gathered} \textit{maximize } \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \\ \textit{subject to } \alpha_i \geq 0 \textit{ for all } i = 1, \ldots, n \\ \textit{and } \sum_{i=1}^{n} \alpha_i y_i = 0. \end{gathered}$$

From an optimal vector α^* we can calculate the optimal normal vector w^* using equation (5). The optimal offset can be calculated with help of equation (3). Please note, that w is a linear combination of those data points x_i with $\alpha_i \neq 0$. These data points are called *support vectors*, hence the name support vector machine. Only support vectors determine the position and orientation of the separating hyperplane, other data points might as well be omitted during learning. In Figure 2 the support vectors are marked with circles. The number of support vectors is usually much smaller than the total number of data points.

3.2 Non-separable Data

We now consider the case that the given set of data points is not linearly separable. The optimization problem discussed in the previous section would not have a solution since in this case constraint (3) could not be fulfilled for all i. We relax this constraint by introducing positive slack variables $\xi_i, i = 1, \ldots, n$. Constraint (3) becomes

$$\forall i : y_i \left(\langle w, x_i \rangle + b \right) \geq 1 - \xi_i.$$

In order to minimize the number of wrong classifications we introduce a correction term $C \sum_{i=1}^{n} \xi_i$ into the objective function. The optimization problems then becomes

$$\begin{aligned} &\text{minimize } \tfrac{1}{2}||w||^2 + C \sum_{i=1}^{n} \xi_i \\ &\text{subject to } \forall i : y_i \left(\langle w, x_i \rangle + b \right) \geq 1 - \xi_i. \end{aligned}$$

The factor C determines the weight of wrong predictions as part of the objective function. As in the previous section we create the dual form of the Lagrangian. The slacking variables ξ_i vanish and we get the following optimization problem:

Problem 3 *The dual SVM problem for non-separable data is defined as*

$$\begin{aligned} &\textit{maximize } \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \\ &\textit{subject to } 0 \leq \alpha_i \leq C \textit{ for all } i = 1, \ldots, n \\ &\textit{and } \sum_{i=1}^{n} \alpha_i y_i = 0. \end{aligned}$$

It can easily be seen that the only difference to the separable case is the additional upper bound C for all α_i.

3.3 Non-linear Learning with Kernels

The optimization problem described with the equations in Problem 3 will deliver a linear separating hyperplane for arbitrary data sets. The result is optimal in a sense that no other linear function is expected to provide a better classification function on unseen data according to $P(X, Y)$. However, if the data is not linearly separable at all the question arises how the described optimization problem can be generalized to non-linear decision functions. Please note that the data points only appear in the form of dot products

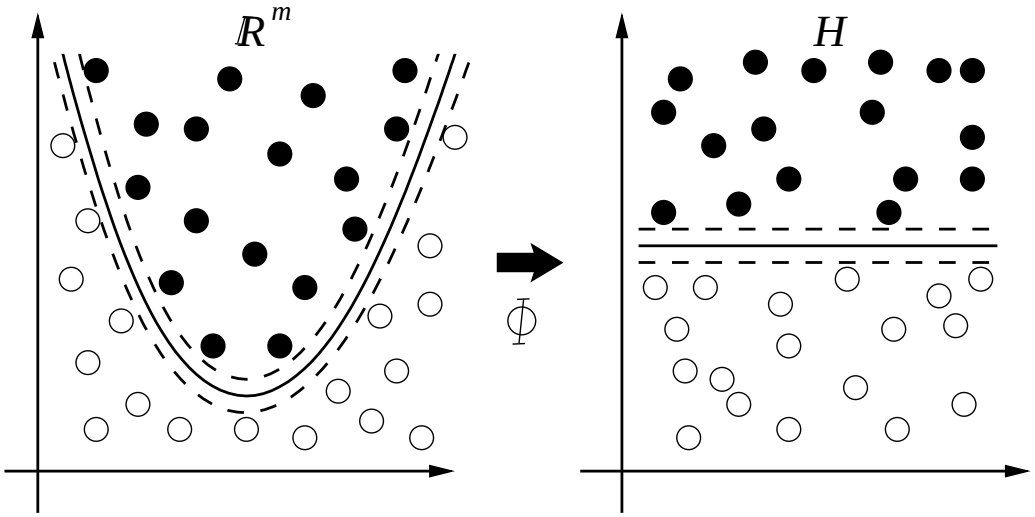

Figure 3: After the transformation of all data points into the feature space H the non-linear separation problem can be solved with a linear separation algorithm. In this case a transformation in the space of polynomials with degree 2 was chosen.

$\langle x_i, x_j \rangle$. A possible interpretation of this dot product is the similarity of these data points in the input space $\mathbb{R}^m$. Now consider a mapping $\Phi : \mathbb{R}^m \to H$ into some other Euclidean space H (called *feature space*) which might be performed before the dot product is calculated. The optimization would depend on dot products in this new space H, i. e. on functions of the form $\langle \Phi(x_i), \Phi(x_j) \rangle$. A function $k : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ with the characteristic

$$k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$$

is called *kernel function* or *kernel*. Figure 3 gives a rough idea how transforming the data points can help to solve non-linear problems with the optimization in a (higher dimensional) space where the points can be linearly separated.

A fascinating property of kernels is that for some mappings Φ a kernel k exists which can be calculated without actually performing Φ. Since often the dimension of H is greater than the dimension m of the input space and H sometimes is even infinite dimensional, the usage of such kernels is a very efficient way to introduce non-linear decision functions into large margin approaches. Prominent examples for such efficient non-linear kernels are polynomial kernels with degree d

$$k(x_i, x_j) = (\kappa \langle x_i, x_j \rangle + \delta)^d,$$

radial basis function kernels (RBF kernels)

$$k(x_i, x_j) = e^{-\frac{||x_i - x_j||^2}{2\sigma^2}}$$

for a $\sigma > 0$, and the sigmoid kernel

$$k(x_i, x_j) = \tanh(\kappa \langle x_i, x_j \rangle - \delta)$$

which can be used to simulate a neural network. κ and δ are scaling and shifting parameters. Since the RBF kernel is easy interpretable and often yields good prediction performance, it is used in a wide range of applications. We will also use the RBF kernel for our experiments described in section 6 in order to demonstrate the learning ability of the proposed evolutionary SVM.

We replace the dot product in the objective function by kernel functions and achieve the final optimization problem for finding a non-linear separation for non-separable data points:

Problem 4 (Final SVM Problem) *The dual SVM problem for non-linear hyperplanes for non-separable data is defined as*

$$\begin{aligned} &\textit{maximize } \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j k(x_i, x_j) \\ &\textit{subject to } 0 \leq \alpha_i \leq C \textit{ for all } i = 1, \ldots, n \\ &\textit{and } \sum_{i=1}^{n} \alpha_i y_i = 0. \end{aligned}$$

4 Learning with Non-Positive Semidefinite Kernels

It has been shown for positive (semi-)definite kernel functions k, i. e. its kernel matrix K is positive definite, that the objective function is concave [3].

Definition 5 *Let X be a set of items. A kernel function k with kernel matrix entries $K_{ij} = k(x_i, x_j)$ is called* POSITIVE SEMIDEFINITE *if the following applies*

$$c^* K c \geq 0 \textit{ for all } c \in \mathbb{C}^n$$

where c^* is the conjugate transpose of c.

The kernel satisfies Mercer's condition in this case [18] and can be seen as dot product in some Hilbert space (this space is usually referred to as *Kernel Reproducing Hilbert Space* (RKHS) [29], see below). If the objective function is concave, it has a global unique maximum which usually can be found by means of a gradient descent. However, in some cases a specialized kernel function must be used to measure the similarity between data points which is not positive definite, sometimes not even positive semidefinite [15]. While positive definite kernels – just as the regular dot product – resemble a similarity measure, these non-positive semidefinite kernels (or indefinite kernels) can be considered as a (partial) distance measure. For such non-positive semidefinite (non-PSD) kernel functions the usual quadratic programming approaches might not be able to find a global maximum in a feasible time since the optimization problem is no longer concave.

One may ask why a solution for non-positive semidefinite kernels would be interesting at all. There are several reasons for studying the effect of non-PSD kernel functions on the optimization problem[3]. First, the test for Mercer's condition can be a challenging task which often cannot be solved by a practitioner. Second, some kernel functions are interesting in spite that it can be shown that they are not positive semidefinite, e.g. the sigmoid kernel function $k(x_i, x_j) = \tanh(\kappa \langle x_i, x_j \rangle - \delta)$ of neural networks or a fractional power polynomial. Third, promising empirical results were reported for such non-PSD or indefinite kernels [16]. Finally, several approaches of learning the kernel function were proposed where the result not necessarily must again be positive semidefinite even if only definite kernel functions were used as base functions [17].

Before we discuss former approaches to learn SVM functions for such non-PSD kernels, we state some of the most important non-positive semidefinite kernel functions for two instances x_i and x_j:

Epanechnikov:

$$\left(1 - \frac{||x_i - x_j||^2}{\sigma_E}\right)^d \text{ for } \frac{||x_i - x_j||^2}{\sigma_E} \leq 1$$

Gaussian Combination:

$$\exp\left(\frac{-||x_i - x_j||^2}{\sigma_{gc1}}\right) + \exp\left(\frac{-||x_i - x_j||^2}{\sigma_{gc2}}\right) - \exp\left(\frac{-||x_i - x_j||^2}{\sigma_{gc3}}\right)$$

Multiquadric:

$$\sqrt{\frac{||x_i - x_j||^2}{\sigma_M} + c^2}$$

[3]For a deeper discussion of the applications of non-PSD kernels see [23].

4.1 Learning in Reproducing Kernel Hilbert Spaces

In this section, we will give a short discussion about the connection between regularization and the feature space induced by the kernel. As we have seen before, the key idea of regularization is to restrict the function class f of possible minimizers of the empirical risk such that f becomes a compact set. In case of kernel based learners, we have to consider the space into which the function Φ of the kernel function maps the data points. This feature space is called *Reproducing Kernel Hilbert Space* (RKHS) and is defined as follows:

Definition 6 *Let X be a non-empty set and H be a Hilbert space of functions $f : X \to \mathbb{R}$ and let k be a positive semidefinite kernel function. If the following does hold*

1. $\langle f, k(x, \cdot)\rangle = f(x)$ *for all* f *in*H
2. $H = \overline{span\{k(x, \cdot)|x \in X\}}$ *where* $\overline{X}$ *is the completion of the set* X

then H is called a Reproducing Kernel Hilbert Space.

The function f is a projection on the kernel functions of x, hence we can say that the function can be *reproduced* by the kernel functions which explains the name. The importance of RKHS lies in the following theorem [14] which is known as Representer Theorem:

Theorem 1 (Representer Theorem) *Let H be a RKHS with kernel k. Denote by Ω a strictly monotonic increasing function, by X a set, and by L an arbitrary loss function. Then each minimizer $f \in H$ of the regularized risk admits a representation of the form*

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x).$$

The significance of this theorem is that although we might be trying to solve an optimization problem in an infinite-dimensional space H it states that the solution lies in the span of m kernels, in particular those which are centered on the training points. We have already seen that these kernel extensions correspond to the support vectors of support vector machines, i.e. the training points yielding $\alpha_i \neq 0$. The complete learning problem is hence formalized as a minimization over a class of functions defined by the RKHS corresponding to a certain kernel function. This is motivated by the fact that in a RKHS the minimization of a regularized loss functional can be seen as a projection problem [29].

4.2 Learning in Reproducing Kernel Kreĭn Spaces

The minimization problem in RKHS can be efficiently solved by transforming the constrained optimization problem into its dual Lagrangian form. However, in Definition 6 of the RKHS a positive definite kernel function is used. Therefore, we cannot simply transfer this minimization idea to the case of non-PSD or indefinite kernel functions.

In [23] the theoretical foundation for many indefinite kernel methods is discussed. Instead of associating these kernel functions with a RKHS, a generalized type of functional space must be used, namely *Reproducing Kernel Kreĭn Spaces* (RKKS). Learning in these spaces share many of the properties of learning in RKHS, such as orthogonality and projection. Since the kernels are indefinite, the loss over this functional space is no longer just minimized but stabilized. This is a direct consequence of the fact that the dot product defined in Kreĭn spaces no longer must be positive. A Kreĭn space is defined as follows:

Definition 7 *Let K be a vector space and $\langle \cdot, \cdot \rangle_K$ an associated dot product. K is called* Kreĭn space *if there exist two Hilbert spaces H_+, H_- spanning K such that*

- *All $f \in K$ can be decomposed into $f = f_+ + f_-$, where $f_+ \in H_+$ and $f_- \in H_-$ and*
- $\forall f, g \in K: \quad \langle f, g \rangle_K = \langle f_+, g_+ \rangle_{H_+} - \langle f_-, g_- \rangle_{H_-}$.

In analogy to the RKHS defined above we can also define Reproducing Kernel Kreĭn Spaces (RKKS) depending on arbitrary kernel functions including indefinite ones [1]. The analysis of the learning problem in a RKKS gives a similar Representer Theorem as the one stated above [23]. The main difference is that the problem of minimizing a regularized risk functional becomes one of finding the stationary point of a similar functional. Moreover, the solution need not to be unique. The generic formulation is given as follows:

Theorem 2 (RKKS Representer Theorem) *Let K be an RKKS with kernel k. Denote by L a loss function, by Ω a strictly monotonic functional, and by $C\{f, X\}$ a continuous functional imposing a set of constraints on f. Then if the optimization problem*

$$\begin{aligned} &\textit{stabilize } L(y, f(x, \gamma)) + \Omega\left(\langle f, f \rangle_K\right) \\ &\textit{subject to } C\{f, X\} \leq d \end{aligned}$$

has a saddle point f, it admits an expansion of the form

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x).$$

In contrast to the optimization in RKHS this optimization does unfortunately not allow the transformation into the dual problem. Therefore, in each optimization iteration one must recalculate all kernel calculations and dot products during the calculation of the loss function anew. Hence, the RKKS optimization problem might be stated as

$$\begin{aligned} &\text{stabilize } {}_{f \in K} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 \\ &\text{subject to } f \in L \subset span\{\alpha_i k(x_i, \cdot)\} \end{aligned}$$

where L is a subspace of the complete search space [23]. If this problem should be solved, it can clearly be seen that for non-PSD or indefinite kernels we have to deal with the original constrained optimization problem. This problem cannot be solved as efficiently as the dual form known from positive definite SVM. Moreover, the solution need not to be unique any more [23]. SVM based on learning in Kreĭn spaces are therefore hardly feasible for real-world problems.

4.3 Learning with Relevance Vector Machines

Since learning in Kreĭn spaces is much harder than learning in Hilbert spaces, we will discuss an alternative approach derived from sparse Bayesian learning in this section. The *Relevance Vector Machine* (RVM) [35] produces sparse solutions using an improper hierarchical prior and optimizing over hyper parameters. One might define these hyper parameters as the parameters of a Gaussian Process' covariance function. The main purpose of RVM is the direct prediction of probabilistic values instead of crisp classifications. For SVM, complex post processing steps like Platt's scaling [24] must be performed in order to derive probabilistic predictions. Even then, examples could be constructed where such post processing approaches will fail. In contrast, the RVM is based on a probabilistic framework which directly allows to predict the correct values. During the last years, additional research was done to further improve these probabilistic RVM predictions [25].

The main advantage to us, however, is the fact that Relevance Vector Machines depend on basis functions which do not need to fulfill Mercer's condition, hence they do not need to be positive semidefinite.

The RVM is, like the SVM, a sparse linear model, i. e. a linear combination of a set of basis functions $\Phi_i(x)$:

$$f(x) = \sum_{i=1}^{n} \alpha_i \Phi_i(x).$$

The prior on the weights is independent Gaussian, $p(\alpha|A) \sim N(0, A^{-1})$, with separate precision hyper parameters $A = \text{diag}[a_1, \ldots, a_n]$. The output noise is assumed to be zero mean i.i.d. Gaussian of variance σ^2 such that $p(y|X, \alpha, \sigma^2) \sim N(f, \sigma^2 I)$. Learning is achieved by maximizing the marginal likelihood

$$p(y|X, A, \sigma^2) = \int p(y|X, \alpha, \sigma^2) p(\alpha|A) d\alpha.$$

Sparsity is achieved since most of the parameters a_i usually become infinite during the optimization. The corresponding basis functions are then removed. The remaining basis functions on training vectors are called *relevance vectors*. Please refer to [35] for more details.

It should be pointed out that it is a common choice to use the same basis function $\Phi(x)$ for all training examples, often a RBF kernel with fixed parameter σ. There is, however, no need for such a fixed basis function and indeed it is even possible to use basis functions which would not lead to a positive semidefinite kernel matrix. Therefore, we use a RVM in our comparison experiments for non-positive semidefinite kernel functions.

5 Evolutionary Computation for Large Margin Optimization

Since traditional SVM are not able to optimize for non-positive semidefinite kernel functions and RVM are not feasible for real-world problems, it is a very appealing idea to replace the usual quadratic programming approaches by an *evolution strategies* (ES) optimization [2]. The used optimization problem is the dual problem for non-linear separation of non-separable data developed in the last sections (Problem 4). Of course it would also be possible to directly optimize the original form of our optimization problem depicted in Problem 1. That is, we could directly optimize the weight vector w and the offset b. As mentioned before, there are two drawbacks: first, the costs of calculating the fitness function would be much higher for the original optimization problem since the fulfillment of all n constraints must be recalculated for each new hyperplane. It is a lot easier to check if all $0 \leq \alpha_i \leq C$ apply. Instead of this we would have to apply the mapping Φ explicitely for each training

example (which would not be possible for infinite mappings like the mapping of the RBF kernel) and we would have to calculate the dot product in $\forall i : y_i \left(\langle w, x_i \rangle + b\right) \geq 1$ from equation (3) each iteration anew. Second, it would not be possible to allow non-linear learning with efficient kernel functions in the original formulation of the problem. Finally, the kernel matrix K with $K_{ij} = k\left(x_i, x_j\right)$ can be calculated beforehand and the training data is never used during optimization again. This further reduces the needed runtime for optimization since the kernel matrix calculation is done only once and no dot product calculations are necessary during the optimization.

This is a nice example for a case, where transforming the objective function beforehand is both more efficient and allows enhancements which would not have been possible before. Transformations of the fitness functions became a very interesting topic recently in evolutionary algorithms research [33].

5.1 evoSVM

We developed a support vector machine based on evolution strategies optimization (evoSVM). Individuals are the real-valued vectors α and mutation is performed by adding a Gaussian distributed random variable with standard deviation $C/10$. In addition, a variance adaptation is conducted during optimization (1/5 rule [26]). Crossover probability is high (0.9). We use tournament selection with a tournament size of 0.25 multiplied by the population size. The initial individuals are random vectors with $0 \leq \alpha_i \leq C$. The maximum number of generations is 10000 and the optimization is terminated if no improvement occurred during the last 10 generations. The population size is 10. In addition to the evoSVM described here, we also tried different mutation schemes and a support vector machine based on Particle Swarm Optimization [13]. Please refer to [19] for further information on the different results.

6 Experiments and Results

In this section we try to evaluate the proposed evolutionary optimization SVM. We compare our implementation to the quadratic programming approaches usually applied to large margin problems. The experiments demonstrate the competitiveness in terms of the original optimization problem, the classification error minimization, the runtime, and the robustness.

In order to compare the evolutionary SVM described in this paper with standard SVM implementations, we also applied two other SVM on all data

sets. Both SVM use a slightly different optimization technique based on quadratic programming. The used implementations were *mySVM* [28] and *LibSVM* [5]. The latter is an adaptation of the widely used SVM^{light} [12]. Especially for the non-PSD case we also applied a RVM learner [35] on all data sets which should provide better results for this type of kernel functions. All experiments were performed with the machine learning environment YALE [20][4].

6.1 Data Sets

We apply the discussed evoSVM as well as the other SVM implementations on two synthetical and six real-world benchmark data sets. The data set Spiral consists of two intertwingling spirals of different classes. For checkerboard, the data set consists of two classes layed out in a 8×8 checkerboard. In addition, we use six benchmark data sets from the UCI machine learning repository [21] and the StatLib data set library [32], because they already define a binary classification task, consist of real-valued numbers only and do not contain missing values. Therefore, we did not need to perform additional preprocessing steps which might introduce some bias. The properties of all data sets are summarized in Table 1. The default error corresponds to the error a lazy default classifier would make by always predicting the major class. Classifiers must produce lower error rates in order to learn at all instead of just guessing.

We use a RBF kernel for all SVM and determine the best parameter value for σ with a grid search parameter optimization for mySVM. This ensures a fair comparison since the parameter is not optimized for the evolutionary SVM. Possible parameters were 0.001, 0.01, 0.1, 1 and 10. The optimal value for each data set is also given in Table 1. For the non-PSD experiments we used an Epanechnikov kernel function with parameters σ_E and d. The optimized values are also given in Table 1.

6.2 Comparison for the Objective Function

The first question is if the evolutionary optimization approach is able to deliver comparable results with respect to the objective function, i.e. the dual optimization problem 4. We applied all SVM implementations on all data sets and calculated the value for the objective function.

In order to determine the objective function values of all methods we perform a k-fold cross validation. That means that the data set T is divided

[4] `http://yale.sf.net/`

Data set	n	m	**Source**	σ	σ_E	d	**Default**
Spiral	500	2	Synthetical	1.000	11.40	8.80	50.00
Checkerboard	300	2	Synthetical	1.000	9.75	3.00	50.00
Liver	346	6	UCI	0.010	218.61	4.74	42.03
Sonar	208	60	UCI	1.000	5.23	9.00	46.62
Diabetes	768	8	UCI	0.001	998.99	2.56	34.89
Lawsuit	264	4	StatLib	0.010	195.56	8.30	7.17
Lupus	87	3	StatLib	0.001	896.28	6.27	40.00
Crabs	200	7	StatLib	0.100	29.37	2.61	50.00

Table 1: The evaluation data sets. n is the number of data points, m is the dimension of the input space. The kernel parameters σ, σ_E, and d were optimized for the comparison SVM learner *mySVM*. The last column contains the default error, i. e. the error for always predicting the major class.

into k disjoint subsets T_i. For each $i \in \{1, \ldots, k\}$ we use $T \backslash T_i$ as training set and the remaining subset T_i as test set. If F_i is the value for the objective function on the training set $T \backslash T_i$ we calculate the average value

$$F = \frac{1}{k} \sum_{i=1}^{k} \frac{F_i}{|T_i|} \tag{7}$$

over all training sets in order to measure the performance and a standard deviation. In our experiments we choose $k = 20$, i. e. for each method the average and standard deviation of 20 different runs is reported.

Table 2 shows the results. It can clearly be seen that for all data sets the evoSVM approach delivers statistically significant higher values than the other SVM approaches in comparable time.

6.3 Comparison for Positive Kernels

In this section we examine the generalization performance of all SVM implementations for a regular positive semidefinite kernel function (a radial basis function kernel).

We again use k-fold cross validation for the calculation of the generalization error. This time, we calculate E_i as the number of wrong predictions on test set T_i which leads to the average classification error of

$$E = \frac{1}{k} \sum_{i=1}^{k} \frac{E_i}{|T_i|} \tag{8}$$

Data set	**Algorithm**	**Objective Function**	**T[s]**
Spiral	evoSVM	99.183 ± 5.867	11
	mySVM	-283.699 ± 7.208	6
	LibSVM	-382.427 ± 12.295	7
Checkerboard	evoSVM	94.036 ± 1.419	4
	mySVM	-114.928 ± 1.923	2
	LibSVM	-127.462 ± 1.595	3
Liver	evoSVM	103.744 ± 7.000	5
	mySVM	-1301.563 ± 84.893	3
	LibSVM	-1640.546 ± 80.228	3
Sonar	evoSVM	48.436 ± 2.937	3
	mySVM	-558.333 ± 31.249	2
	LibSVM	-491.039 ± 26.196	2
Diabetes	evoSVM	209.491 ± 14.003	10
	mySVM	-90.856 ± 3.566	8
	LibSVM	-108.242 ± 3.886	7
Lawsuit	evoSVM	80.024 ± 18.623	3
	mySVM	-8790.429 ± 308.996	1
	LibSVM	-9061.420 ± 303.227	1
Lupus	evoSVM	29.074 ± 2.582	1
	mySVM	-603.404 ± 52.356	1
	LibSVM	-504.564 ± 41.593	1
Crabs	evoSVM	32.413 ± 1.231	2
	mySVM	-90.856 ± 3.566	1
	LibSVM	-108.242 ± 3.886	1

Table 2: Comparison of the different implementations with regard to the objective function (the higher the better). The results are obtained by a 20-fold cross validation, the time is the cumulated time for all runs.

We again use $k = 20$ in our experiments.

Table 3 summarizes the results for $C = 1$. This value corresponds to $1/\left(1 - \sum K_{ii}\right)$ which is a successful heuristic for determining C proposed by [9]. It can clearly be seen that the evoSVM leads to classification errors comparable to those of the quadratic programming counterparts (mySVM and LibSVM).

The reason for slightly higher errors in some of the predictions of the quadratic programming approaches is probably a too aggressive termination criterion. Although this termination behavior further reduces runtime for mySVM and LibSVM, the classification error is often increased. On the other hand, for the cases, where the evoSVM yields higher errors, the reason probably is a higher degree of overfitting due to the better optimization of the dual problem (please refer to Section 6.2). This can easily be augmented by lower values of C.

Please note that the standard deviations of the errors achieved with the evolutionary SVM are similar to the standard deviations achieved with mySVM or LibSVM. We can therefore conclude that the evolutionary optimization is as robust as the quadratic programming approaches and differences mainly derive from different subsets for training and testing due to cross validation instead of the used randomized heuristics.

6.4 Comparison for Non-positive Kernels

We compared the different implementations and a relevance vector machine on all data sets for a non positive kernel function. The Epanechnikov kernel was used for this purpose with kernel parameters as given in Table 1. These parameters are optimized for the SVM implementation mySVM in order to ensure fair comparisons. Table 4 summarizes the results, again for $C = 1$ which is also the heuristically best value for the Epanechnikov kernel.

It should be noticed that the runtime of the Relevance Vector Machine implementation is not feasible for real-world applications. Even on the comparatively small data sets the RVM needs more than 60 days in some of the cases for all of the twenty learning runs. Although very competitive compared to the other SVM approaches, the RVM was unfortunately not able to deliver the best results for any of the data sets.

It can also be noticed that the evoSVM variant frequently outperforms the competitors on all data sets. While the LibSVM was hardly able to generate models better than the default model the mySVM delivers surprisingly good predictions even for the non-PSD kernel function. However, especially for the data sets Spiral, Checkerboard, Liver, Sonar, and Crabs the results of the evoSVM are significantly better than all competitors.

Data set	**Algorithm**	**Error**	**T[s]**
Spiral	evoSVM	16.40 ± 4.54	11
	mySVM	17.20 ± 4.58	6
	LibSVM	17.80 ± 3.94	7
Checkerboard	evoSVM	22.67 ± 4.90	4
	mySVM	24.00 ± 6.29	2
	LibSVM	23.00 ± 5.04	3
Liver	evoSVM	33.92 ± 6.10	5
	mySVM	31.31 ± 5.86	3
	LibSVM	33.33 ± 4.51	3
Sonar	evoSVM	16.40 ± 9.61	3
	mySVM	14.50 ± 9.61	2
	LibSVM	13.98 ± 7.65	2
Diabetes	evoSVM	25.52 ± 4.30	10
	mySVM	23.83 ± 4.46	8
	LibSVM	24.48 ± 4.81	7
Lawsuit	evoSVM	31.00 ± 11.08	3
	mySVM	29.50 ± 5.56	1
	LibSVM	36.72 ± 2.01	1
Lupus	evoSVM	23.89 ± 14.22	1
	mySVM	24.17 ± 12.87	1
	LibSVM	24.17 ± 12.87	1
Crabs	evoSVM	3.50 ± 3.91	2
	mySVM	3.00 ± 2.45	1
	LibSVM	3.50 ± 3.91	1

Table 3: Comparison of the different implementations with regard to the classification error (the lower the better). The results are obtained by a 20-fold cross validation, the time is the cumulated time for all runs.

Data set	**Algorithm**	**Error**	**T[s]**
Spiral	evoSVM	16.00 ± 4.38	4
	mySVM	46.20 ± 6.56	6
	LibSVM	51.00 ± 1.00	2
	RVM	19.80 ± 3.16	2942468
Checkerboard	evoSVM	23.00 ± 3.56	1
	mySVM	40.33 ± 4.99	1
	LibSVM	45.33 ± 1.63	1
	RVM	38.00 ± 7.48	336131
Liver	evoSVM	31.29 ± 5.90	3
	mySVM	38.32 ± 7.84	1
	LibSVM	42.03 ± 1.46	1
	RVM	40.58 ± 1.75	203093
Sonar	evoSVM	15.40 ± 3.66	3
	mySVM	46.62 ± 1.62	2
	LibSVM	46.62 ± 1.62	2
	RVM	29.79 ± 7.29	105844
Diabetes	evoSVM	32.68 ± 4.77	3
	mySVM	32.54 ± 2.82	4
	LibSVM	34.89 ± 0.34	6
	RVM	32.76 ± 1.89	5702491
Lawsuit	evoSVM	29.89 ± 10.71	2
	mySVM	30.93 ± 10.66	2
	LibSVM	36.72 ± 2.01	2
	RVM	37.89 ± 3.83	106697
Lupus	evoSVM	31.81 ± 11.64	2
	mySVM	34.44 ± 18.51	2
	LibSVM	40.00 ± 6.33	1
	RVM	33.06 ± 10.07	5805
Crabs	evoSVM	4.00 ± 4.90	1
	mySVM	11.00 ± 7.35	1
	LibSVM	50.00 ± 0.00	1
	RVM	13.50 ± 7.43	150324

Table 4: Comparison of the different implementations with regard to the classification error (the lower the better) for a non-positive semidefinite kernel function (Epanechnikov). The results are obtained by a 20-fold cross validation, the time is the cumulated time for all runs.

The following is a brief summarization of the empirical results:

- The evolutionary optimization scheme frequently yields better values for the original objective function.
- The evolutionary optimization scheme lead to comparable results in similar time with respect to classification performance.
- For the case of non-positive semidefinite kernel functions, the evolutionary optimization scheme clearly outperform both traditional support vector machines and a relevance vector machine.

7 Conclusion

In this paper, we connected evolutionary computation with statistical learning theory. The idea of large margin methods was very successful in many applications from machine learning and data mining. We used the most prominent representative of this paradigm, namely Support Vector Machines, and employed evolution strategies in order to solve the constrained optimization problem at hand.

An interesting property of large margin methods is that the runtime for fitness evaluation is reduced by transforming the problem into the dual problem. In our case, the algorithm is both faster and provides space for other improvements like incorporating a kernel function for non-linear classification tasks. This is a nice example how a transformation into the dual optimization problem can also be exploited by evolutionary algorithms.

With respect to the objective function, the evolutionary SVM always outperform their quadratic counterparts. With respect to the generalization ability (prediction accuracy), this leads to at least as accurate results as for their competitors. We can conclude that evolutionary algorithms proved as reliable as other optimization schemes for this type of problems. Beside the inherent advantages of evolutionary algorithms (e. g. parallelization, multi-objective optimization of training error and capacity) it is now also possible to employ non-positive semidefinite or indefinite kernel functions which would lead to unsolvable problems for other optimization techniques. As the experiments have shown, a SVM based on evolutionary computation is the first practical solution for this type of problem.

8 Acknowledgments

This work was supported by the *Deutsche Forschungsgemeinschaft (DFG)* within the *Collaborative Research Center "Reduction of Complexity for Multivariate Data Structures"*. The author would also like to thank Stefan Rüping for fruitful discussions and the implementation of his SVM.

References

[1] D. Alpay. The schur algorithm, reproducing kernel spaces and system theory. In *SMF/AMS Texts and Monographs*, volume 5. SMF, 2001.

[2] H.-G. Beyer and H.-P. Schwefel. Evolution strategies: A comprehensive introduction. *Journal Natural Computing*, 1(1):2–52, 2002.

[3] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[4] G. Camps-Valls, J.D. Martin-Guerrero, J.L. Rojo-Alvarez, and E. Soria-Olivas. Fuzzy sigmoid kernel for support vector classifiers. *Neurocomputing*, 62:501–506, 2004.

[5] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001.

[6] F. Friedrichs and C. Igel. Evolutionary tuning of multiple svm parameters. In *Proc. of the 12th European Symposium on Artificial Neural Networks (ESANN 2004)*, pages 519–524, 2004.

[7] H. Frþhlich, O. Chapelle, and B. Schölkopf. Feature selection for support vector machines using genetic algorithms. *International Journal on Artificial Intelligence Tools*, 13(4):791–800, 2004.

[8] B. Haasdonk. Feature space interpretation of svms with indefinite kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):482–492, 2005.

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer, 2001.

[10] T. Howley and M.G. Madden. The genetic kernel support vector machine: Description and evaluation. *Artificial Intelligence Review*, 2005.

[11] W. James and C. Stein. Estimation with quadratic loss. In *Proceedings of the Fourth Berkeley Symposium on Mathematics, Statistics and Probability*, pages 361–380, 1960.

[12] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11. MIT Press, Cambridge, MA, 1999.

[13] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proc. of the International Conference on Neural Networks*, pages 1942–1948, 1995.

[14] G. S. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33:82–95, 1971.

[15] H.-T. Lin and C.-J. Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods, March 2003.

[16] H.-T. Lin and C.-J. Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods, 2003.

[17] X. Mary. *Hilbertian Subspaces, Subdualities and Applications*. PhD thesis, Institut National des Sciences Appliquees Rouen, 2003.

[18] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society*, A 209:415–446, 1909.

[19] I. Mierswa. Evolutionary learning with kernels: A generic solution for large margin problems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006)*, 2006.

[20] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006)*, 2006.

[21] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[22] C. Ong, X. Mary, S. Canu, and A. J. Smola. Learning with non-positive kernels. In *Proc. of the 21st International Conference on Machine Learning (ICML)*, pages 639–646, 2004.

[23] C. Ong, X. Mary, S. Canu, and A. J. Smola. Learning with non-positive kernels. In *Proceedings of the 21st International Conference on Machine Learning (ICML 2004)*, pages 639–646, 2004.

[24] J. Platt. *Advances in Large Margin Classifiers*, chapter Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. MIT Press, 1999.

[25] C. E. Rasmussen and J. Quinonero-Candela. Healing the relevance vector machine through augmentation. In *Proc. of the 22nd International Conference on Machine learning (ICML 2005)*, pages 689–696. ACM Press, 2005.

[26] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[27] T.P. Runarsson and S. Sigurdsson. Asynchronous parallel evolutionary model selection for support vector machines. *Neural Information Processing*, 3(3):59–67, 2004.

[28] Stefan Rüping. *mySVM Manual*. Universität Dortmund, Lehrstuhl Informatik VIII, 2000. http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/.

[29] B. Schölkopf and A. J. Smola. *Learning with Kernels – Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[30] A. Smola, B. Schölkopf, and K.-R. Müller. General cost functions for support vector regression. In *Proceedings of the 8th International Conference on Artificial Neural Networks*, pages 79–83, 1998.

[31] A. J. Smola, Z. L. Ovari, and R. C. Williamson. Regularization with dot-product kernels. In *Proc. of the Neural Information Processing Systems (NIPS)*, pages 308–314, 2000.

[32] Statlib – datasets archive. http://lib.stat.cmu.edu/datasets/.

[33] T. Storch. On the impact of objective function transformations on evolutionary and black-box algorithms. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 833–840, 2005.

[34] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: A large margin approach. In *Proc. of the International Conference on Machine Learning (ICML)*, 2005.

[35] M. E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

[36] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[37] V. Vapnik and A. Chervonenkis. The necessary and sufficient conditions for consistency in the empirical risk minimization method. *Pattern Recognition and Image Analysis*, 1(3):283–305, 1991.